**Control Flow**

1. **Goto Statements**

In C++, the goto statement is a control flow statement that allows you to transfer control to a labeled statement within the same function. While goto can be a powerful tool, it is often considered bad practice to use it because it can lead to spaghetti code and make the program difficult to understand and maintain. It is generally recommended to use structured control flow statements like if, while, for, and switch instead of goto whenever possible.

The basic syntax of the goto statement is as follows:

goto label\_name;

// ...

label\_name:

statement;

1. **Conditional Statements:**

* **If**
* **If-else**
* **If-else Ladder**
* **Nested-If**
* **Nested if-else**
  1. **If:**

**Syntax of If statement:**

If (condition) {

statement-1;

}

* 1. **If/else:**

**Syntax of If-else statement**

If (condition) {

statement-1;

} else {

statement-2;

}

* 1. **If-else Ladder:**

**Syntax of If-else Ladder**

If (condition 1) {

}

else if (condition 2) {

}

else if (condition 3) {

}

else if (condition 4) {

}

else {

}

* 1. **Nested if**

**Nested if means there is another if statement inside if**

**Syntax if Nested If:**

If (condition 1) {

if (condition 2) {

}

}

* 1. **Nested if else:**

**Syntax of Nested if else:**

if (condition1) {

Code to execute if condition1 is true

if (condition2) {

Code to execute if both condition1 and condition2 are true

} else {

Code to execute if condition1 is true but condition2 is false

}

} else {

Code to execute if condition1 is false

}

1. **Switch Case**

switch (expression) {

case constant1:

Code to execute if expression is equal to constant1

break;

case constant2:

Code to execute if expression is equal to constant2

break;

default:

Code to execute if expression doesn't match any constant

}

1. **Loops**

* **While**
* **Do While**
* **For Loop**
* **Nested Loop**
  + **Nested While**
  + **Nested Do While**
  + **Nested For**
  1. **While:**

**Syntax:**

while (test expression) {

Statements

Increment/decrement.

}

* 1. **Do While**

**Syntax:**

do {

Statements;

Increment/decrement;

}

While (test expression);

* 1. **For Loop:**

**Syntax:**

For (initialization, Boolean expression, update area) {

Statements

}

* 1. **Nested loops:**

Nested loops are loops within loops

**Syntax:**

for (int i=0; condition; increment) {

for (int i=0; condition; increment) {

statement(s);

}

}

**Note:**

All the program related to nested loop are present in the word file, which is present in the Nested Loops folder

1. **Break and Continue Statements**

**Break Statement:**

* The break statement is used to exit a loop or a switch statement prematurely.
* When a break statement is encountered inside a loop, the loop is terminated, and the control flow moves to the statement immediately following the loop.
* In a switch statement, break is used to exit the switch block after a case is executed, preventing fall-through to the next case.

**Example of break in a loop:**

for (int i = 0; i < 5; i++) {

if (i == 3) {

break; // Exit the loop when i reaches 3

}

cout << i << " ";

}

**Example of break in a switch statement:**

int choice = 2;

switch (choice) {

case 1:

cout << "Option 1 selected" << endl;

break;

case 2:

cout << "Option 2 selected" << endl;

break;

default:

cout << "Invalid choice" << endl;

}

**Continue Statement:**

* The continue statement is used to skip the rest of the current iteration of a loop and proceed to the next iteration.
* When a continue statement is encountered inside a loop, the remaining code in the loop body is skipped, and the loop's condition (if any) is checked to determine if the next iteration should begin.

**Example of continue in a loop:**

for (int i = 0; i < 5; i++) {

if (i == 2) {

continue; // Skip iteration when i is 2

}

cout << i << " ";

}